The MIPS Pipelined Processor with Forwarding

Basic Simulations in ModelSim of the Processor with Forwarding and Hazard Detection

Due: June 5, 2016

By: Avik Bag, Daniel Schoepflin

Contents

Overview

In the third portion of this project (Part 2: Forwarding and hazard detection), the goal was to improve upon the pipelined MIPS processor developed in the second part of the project by adding processor hazard detection and forwarding capabilities. The processor developed in this portion of the experiment would have numerous benefits over that which was created in Part 1, since the total number of stall cycles needed when forwarding was enabled was greatly reduced. Additionally, the hazard detection allowed forwarding to function with impunity – even when a memory access instruction was executed – since a stall would be inserted in the case of potential data hazard. Unlike the improvements made from Part 0 to Part 1, which were difficult to visualize on the on the waveforms, the improvements from Part 1 to Part 2 are noticeable and are explored in the "Annotated Result Screenshots and Analysis" section. The design of the processor was based on the diagrams pictured in Figs. 1 and 2 which follow in the "Basic Design Principles" section.

Basic Design Principles

In designing the improvements for this processor, the main consideration was maintaining full functionality of the previous design, including branches, jumps, and read-through writes of the registers. The modularity of the design of the processor in Part 1 allowed for easy extensibility and changes to be made. The first improvement that was made was the addition of the forwarding unit as shown in Fig. 1.

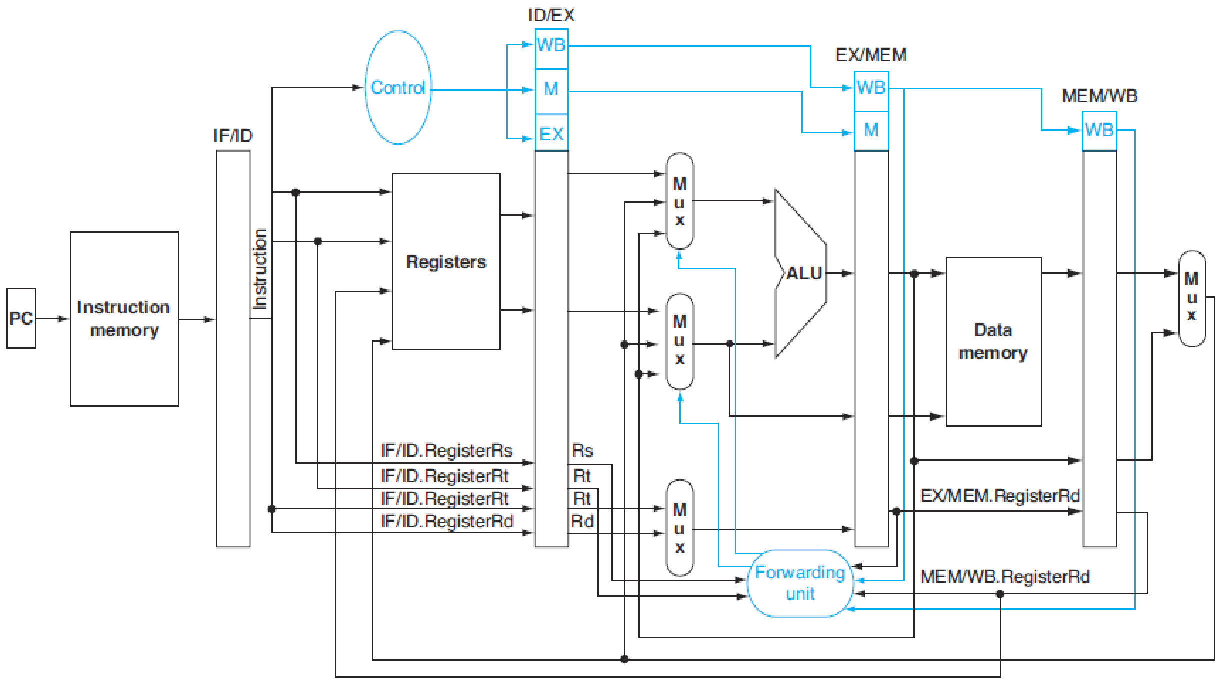


Fig. 1. Diagram taken from Patterson and Hennessy text of the MIPS pipelined processor with a forwarding unit in place.

Note that the diagram above was not directly implemented since it does not preserve the ability to use branch or jump instructions. Instead, the lines from the forwarding unit were added along with its logic and the two input multiplexors were expanded to three input multiplexors so as to allow forwarded data to be used.

The second major improvement that was required was the addition of the hazard detection unit. This unit was added in order to dynamically insert stalls in the case of memory access instructions introducing dependencies. A diagram for the design of this section is shown in Fig. 2 below.

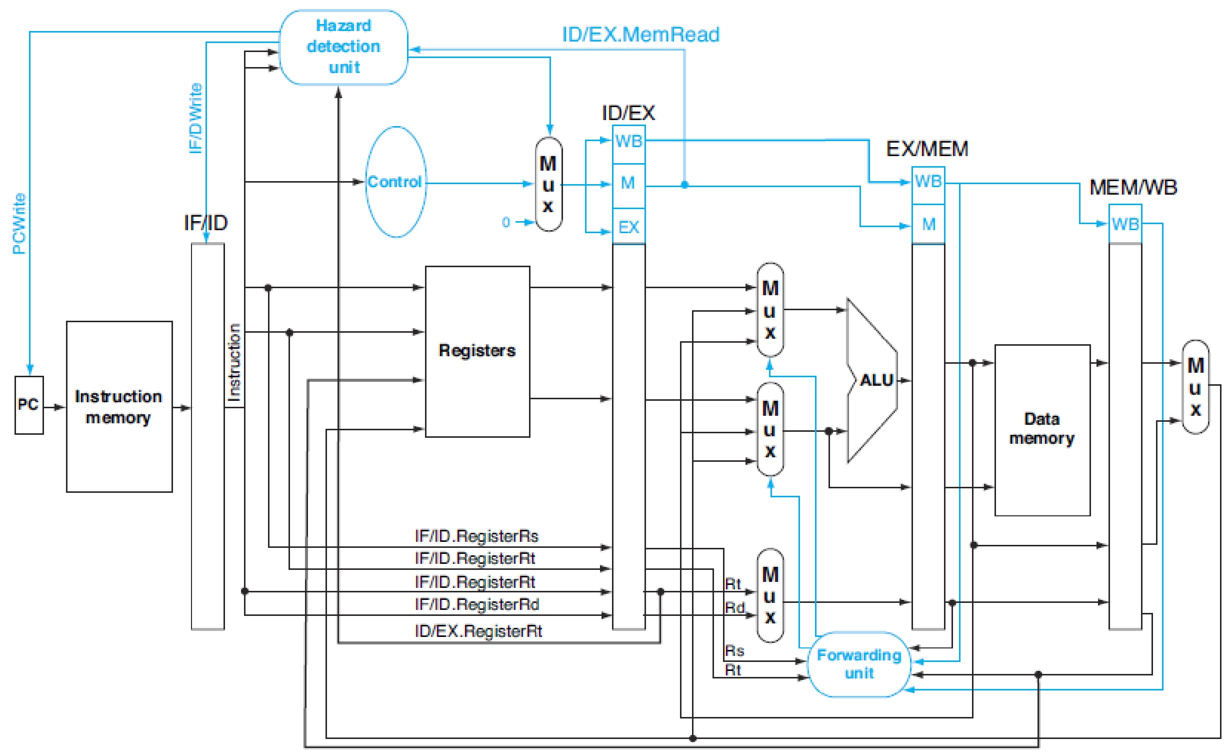
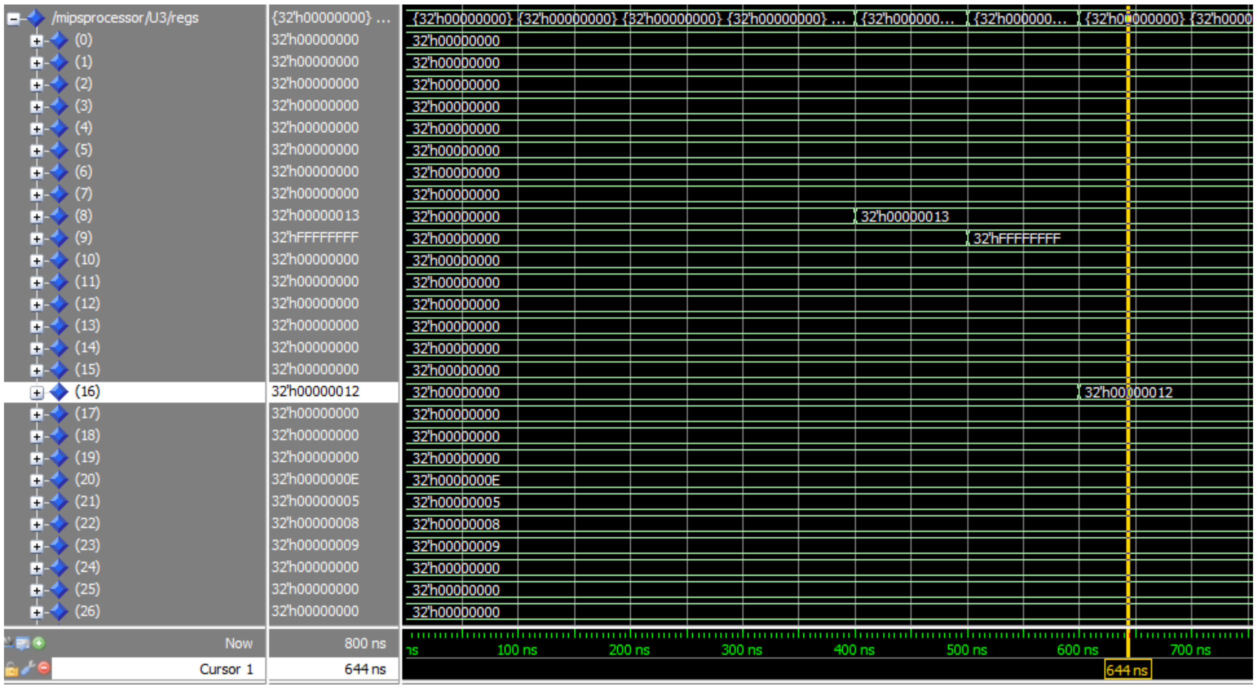


Fig. 2. Diagram taken from Patterson and Hennessy text of the MIPS pipelined processor with a hazard detection unit in place.

Annotated Result Screenshots and Analysis

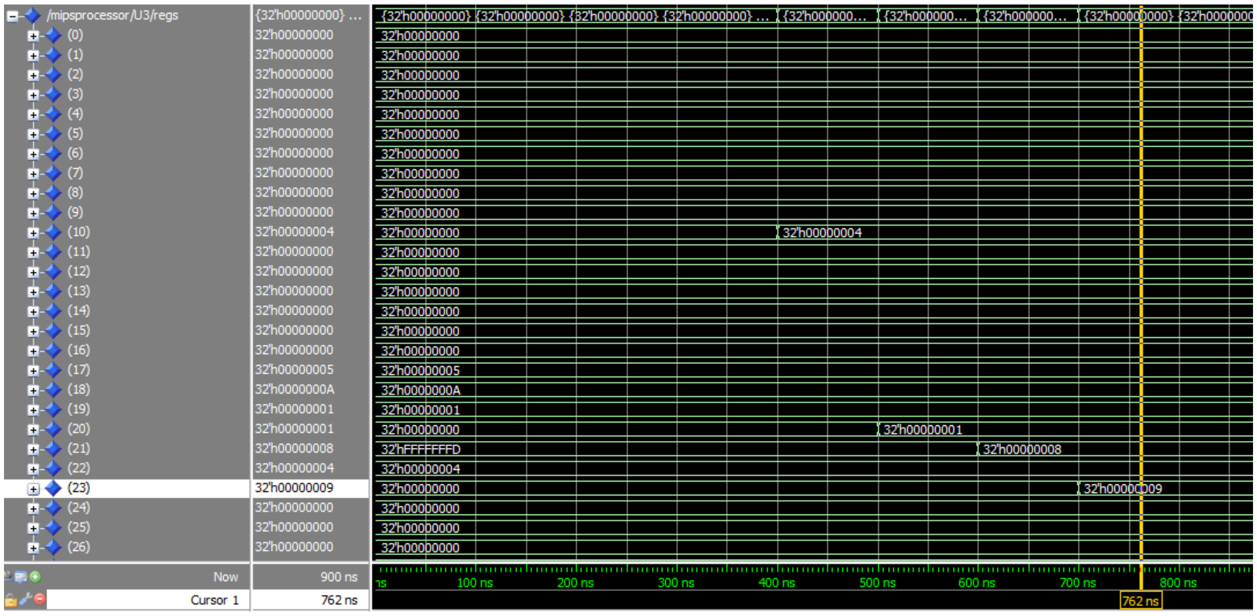
In order to confirm the functionality of the two added components of the processor, sample test code was provided and the processor VHDL code was simulated in ModelSim. The results from each of the four programs are shown in this section and illustrate the correct functionality of the circuit as a whole. In the first program, the data hazards between three instructions were resolved using forwarding. The resulting waveform for Program 1 is shown in Fig. 3.



/mipsprocessor/U3/regs	{32'h00000000} ...
(0)	32'h00000000
(1)	32'h00000000
(2)	32'h00000000
(3)	32'h00000000
(4)	32'h00000000
(5)	32'h00000000
(6)	32'h00000000
(7)	32'h00000000
(8)	32'h00000013
(9)	32'hFFFFFFFF
(10)	32'h00000000
(11)	32'h00000000
(12)	32'h00000000
(13)	32'h00000000
(14)	32'h00000000
(15)	32'h00000000
(16)	32'h00000012
(17)	32'h00000000
(18)	32'h00000000
(19)	32'h00000000
(20)	32'h0000000E
(21)	32'h00000005
(22)	32'h00000008
(23)	32'h00000009
(24)	32'h00000000
(25)	32'h00000000
(26)	32'h00000000
Now	800 ns
Cursor 1	644 ns

Fig. 3. Resulting registers from executing Program 1 using the pipelined processor with forwarding. Notice that $t0 is made 19, $t1 is made -1, and $s0 is made 18 in the appropriate clock cycles.

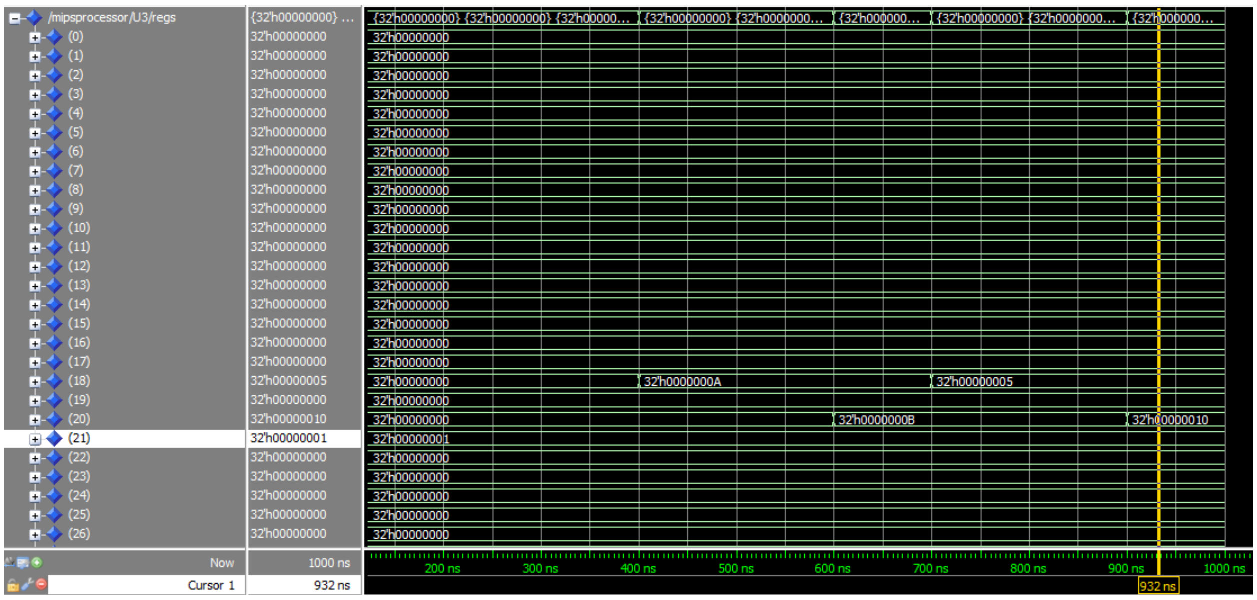
The second program also worked to confirm functionality of the forwarding unit by having multiple dependent instructions. The resulting waveform is shown in Fig. 4 below.



/mipsprocessor/U3/regs	{32'h00000000} ...
(0)	32'h00000000
(1)	32'h00000000
(2)	32'h00000000
(3)	32'h00000000
(4)	32'h00000000
(5)	32'h00000000
(6)	32'h00000000
(7)	32'h00000000
(8)	32'h00000000
(9)	32'h00000000
(10)	32'h00000004
(11)	32'h00000000
(12)	32'h00000000
(13)	32'h00000000
(14)	32'h00000000
(15)	32'h00000000
(16)	32'h00000000
(17)	32'h00000005
(18)	32'h0000000A
(19)	32'h00000001
(20)	32'h00000001
(21)	32'h00000008
(22)	32'h00000004
(23)	32'h00000009
(24)	32'h00000000
(25)	32'h00000000
(26)	32'h00000000
Now	900 ns
Cursor 1	762 ns

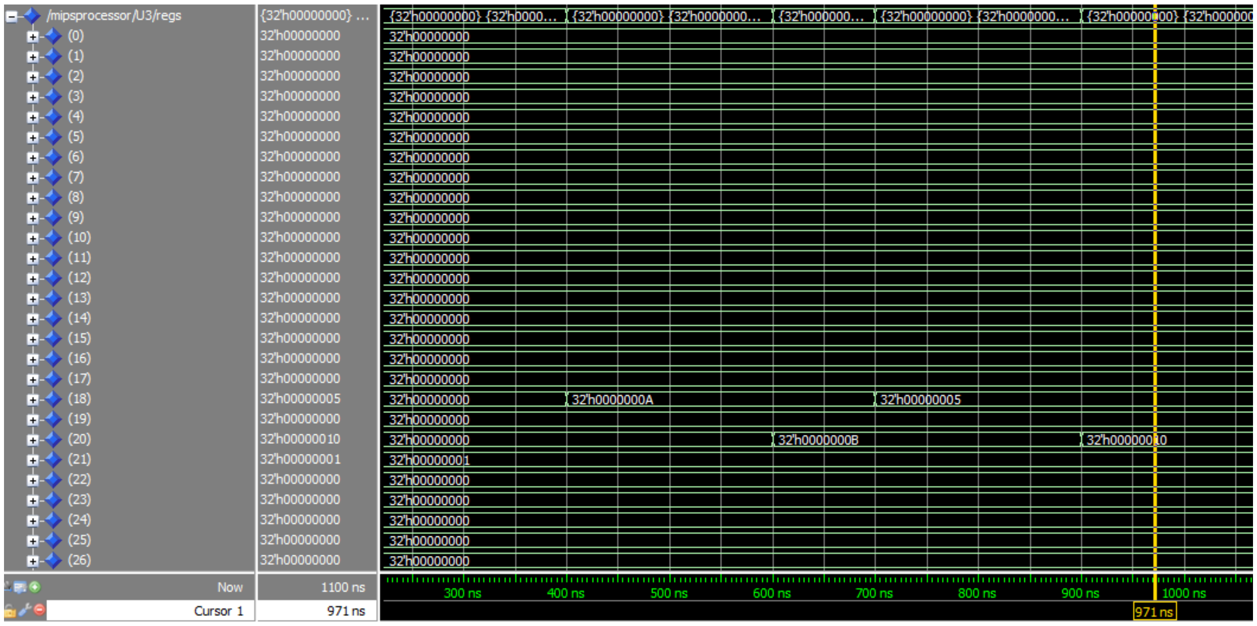
Fig. 4. Resulting registers from executing Program 2 using the pipelined processor with forwarding. Notice that $t2 is set to 4, $s4 is set to 1, $s5 is set to 8, and $s7 is set to 9 in the appropriate cycles.

In both Fig. 3 and Fig. 4, the forwarding is shown to work properly since the values of the registers update based on values calculated in the previous EX stage. This is the simplest kind of forwarding since there is no need to stall execution. The forwarding unit simply determines if the destination register of a previous instruction is a source register for a later instruction and alerts a mux appropriately. This is complicated when a stall is required, as is the case for Programs 3 and 4 which access memory and have dependencies on the values obtained in these stages. Since an instruction in the MEM phase cannot pass the value obtained from data memory to the EX phase of the following instruction (since they occur in the same cycle), another unit is required to identify these situations and prevent the program from proceeding by introducing a stall. Additionally, the program counter must be held at the current value (by subtracting 4 since the incrementing of the PC occurs on every clock cycle, regardless of scenario) so as to “refetch” the instruction which was flushed from the pipeline. The results of these enhancements are shown in Figs. 5, 6, and 7 which show the registers for Program 3 and the registers and data memories from Program 4, respectively.



```
/mipsprocessor/U3/regs  {32'h00000000} ...  {32'h00000000} {32'h00000000} {32'h00000... {32'h00000000} {32'h0000000... {32'h000000... {32'h00000000} {32'h0000000... {32'h000000...
  (0)   32'h00000000   32'h00000000
  (1)   32'h00000000   32'h00000000
  (2)   32'h00000000   32'h00000000
  (3)   32'h00000000   32'h00000000
  (4)   32'h00000000   32'h00000000
  (5)   32'h00000000   32'h00000000
  (6)   32'h00000000   32'h00000000
  (7)   32'h00000000   32'h00000000
  (8)   32'h00000000   32'h00000000
  (9)   32'h00000000   32'h00000000
  (10)  32'h00000000   32'h00000000
  (11)  32'h00000000   32'h00000000
  (12)  32'h00000000   32'h00000000
  (13)  32'h00000000   32'h00000000
  (14)  32'h00000000   32'h00000000
  (15)  32'h00000000   32'h00000000
  (16)  32'h00000000   32'h00000000
  (17)  32'h00000000   32'h00000000
  (18)  32'h00000005   32'h00000000   32'h0000000A   32'h00000005
  (19)  32'h00000000   32'h00000000
  (20)  32'h00000010   32'h00000000   32'h0000000B   32'h00000010
  (21)  32'h00000001   32'h00000001
  (22)  32'h00000000   32'h00000000
  (23)  32'h00000000   32'h00000000
  (24)  32'h00000000   32'h00000000
  (25)  32'h00000000   32'h00000000
  (26)  32'h00000000   32'h00000000
Now       1000 ns     200 ns  300 ns  400 ns  500 ns  600 ns  700 ns  800 ns  900 ns  1000 ns
Cursor 1   932 ns     932 ns
```

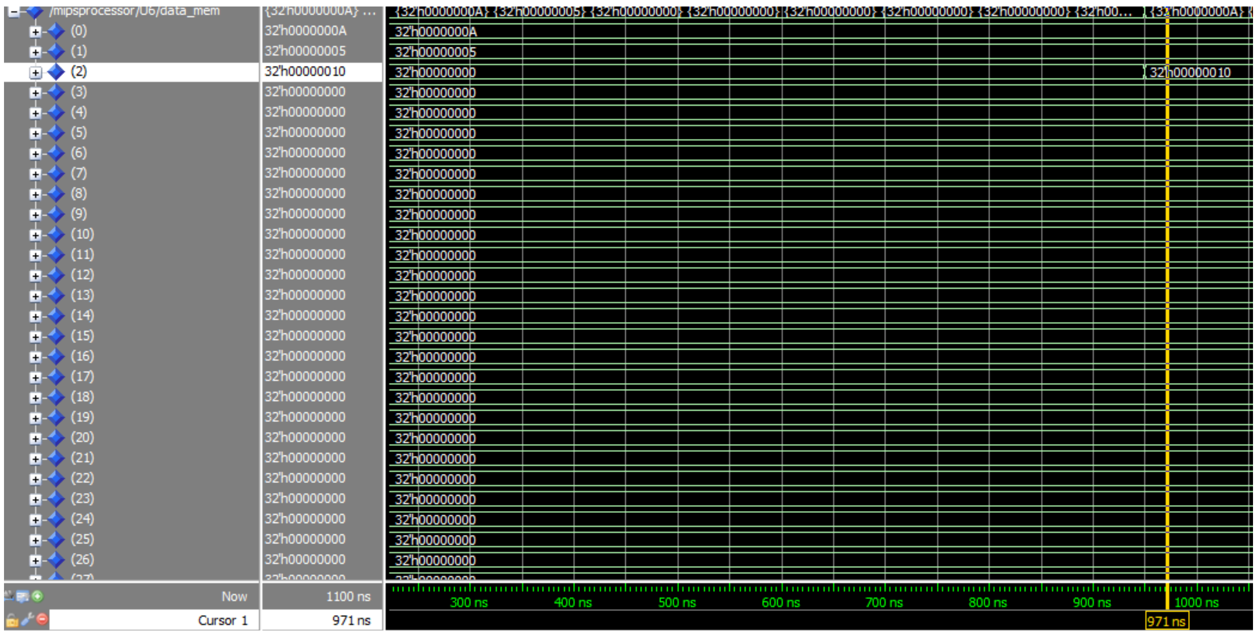
Fig. 5. Resulting registers from executing Program 3 using hazard detection. Notice that the registers update with appropriate values resulting in $s5 containing 16 in clock cycle 10.



```
/mipsprocessor/U3/regs   {32'h00000000} ...
(0)    32'h00000000
(1)    32'h00000000
(2)    32'h00000000
(3)    32'h00000000
(4)    32'h00000000
(5)    32'h00000000
(6)    32'h00000000
(7)    32'h00000000
(8)    32'h00000000
(9)    32'h00000000
(10)   32'h00000000
(11)   32'h00000000
(12)   32'h00000000
(13)   32'h00000000
(14)   32'h00000000
(15)   32'h00000000
(16)   32'h00000000
(17)   32'h00000000
(18)   32'h00000005
(19)   32'h00000000
(20)   32'h00000010
(21)   32'h00000001
(22)   32'h00000000
(23)   32'h00000000
(24)   32'h00000000
(25)   32'h00000000
(26)   32'h00000000
Now       1100 ns
Cursor 1   971 ns
```

{32'h00000000} {32'h0000... {32'h00000000} {32'h0000000... {32'h000000... {32'h00000000} {32'h0000000... {32'h00000000} {32'h000000
32'h00000000
32'h0000000A
32'h00000005
32'h0000000B
32'h00000010
32'h00000001
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1000 ns
971 ns

Fig. 6. Resulting registers from executing Program 4 using hazard detection. Notice that the result is identical to that seen in Program 3 since the only difference between the two programs was an additional store word instruction occurring at the end in Program 4.



```
/mipsprocessor/U6/data_mem   {32'h0000000A} ...
(0)    32'h0000000A
(1)    32'h00000005
(2)    32'h00000010
(3)    32'h00000000
(4)    32'h00000000
(5)    32'h00000000
(6)    32'h00000000
(7)    32'h00000000
(8)    32'h00000000
(9)    32'h00000000
(10)   32'h00000000
(11)   32'h00000000
(12)   32'h00000000
(13)   32'h00000000
(14)   32'h00000000
(15)   32'h00000000
(16)   32'h00000000
(17)   32'h00000000
(18)   32'h00000000
(19)   32'h00000000
(20)   32'h00000000
(21)   32'h00000000
(22)   32'h00000000
(23)   32'h00000000
(24)   32'h00000000
(25)   32'h00000000
(26)   32'h00000000
Now       1100 ns
Cursor 1   971 ns
```

{32'h0000000A} {32'h00000005} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00... {32'h0000000A}
32'h0000000A
32'h00000005
32'h00000000
32'h00000010
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1000 ns
971 ns

Fig. 7. Resulting data memory from executing Program 4 using hazard detection. Notice that the data memory at array index 2 (that is, Mem[8-11]) is set to 16, the desired value in clock cycle 10.

Further Enhancements and Future Work

Unfortunately, even with the addition of forwarding and hazard detection, there are still some unavoidable stalls in the system. This occurs due to the hazard detection working to resolve data dependencies between memory accessing instructions and register instructions which rely on the values obtained therein. Further, since the stall is mandatory due to the hazard detection unit, there is no way to “make use” of the stall as was possible in the case of the branch delay slot. Therefore, in the future it would be useful to implement a means of dynamically executing the instructions out-of-order when possible (through the means of a scoreboard, for example). However, this improvement was well outside the scope of the current task, and would thus be an enhancement attempted further in the future.

Conclusion

This project largely centered on the addition of two new modules and their wiring into an already somewhat complicated schematic. Interestingly, the development of the modules was not tremendously difficult, as their behavior is well documented and precisely noted. The most difficult part was, in fact, ensuring that all signals were correctly assigned by each module and piped to each buffer or processor component as necessary. Furthermore, ensuring that the number of stalls was minimized to one proved to be simple to execute but conceptually a challenge, since the original thought was to completely flush the pipeline. This project was quite illuminating on the whole and demonstrated the value of forwarding and hazard detection quite clearly since the enhancements made dramatically improved performance.